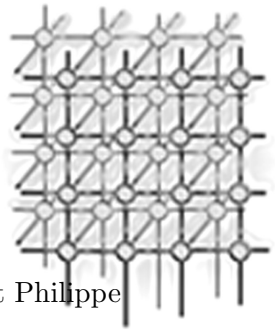
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

DTM: a service for managing data persistency and data replication in NES environments



Bruno Del Fabbro, David Laiymani, Jean-Marc Nicod, Laurent Philippe

Laboratoire d'Informatique de l'université de Franche-Comté
16 route de Gray, 25030 Besançon Cedex - France

SUMMARY

Network-Enabled Servers (NES) environments are valuable candidates to provide simple computing grid access. These environments allow transparent access to a set of computational servers via Remote Procedure Call mechanisms. In this context, a challenge is to increase performances by decreasing data traffic. This paper presents DTM (Data Tree Manager) a data management service for NES environments. Based on the notions of data persistency and data replication, DTM proposes a set of efficient policies which minimise computation times by decreasing data transfers between the clients and the platform. From the end-user point of view, DTM is accessible through a simple and transparent API. In the remainder, we describe DTM and its implementation in the DIET platform. We also present a set of experimental results which exhibit the feasibility and the efficiency of our approach.

KEY WORDS: Grid computing, data management, persistency, replication, DIET

1. Introduction

Grids environments connect a large number of distributed resources which produce and require a large amount of data [1, 2]. In this context, data management is now a key issue for Grid environments. A lot of projects [3, 4] propose data management solutions. They aim at providing reliable access to data for users that have no specific information about data characteristics such as location, name or attributes. These works have resulted in the notion of data virtualization which consists in the separation of the data physical location from their logical view. These solutions are mainly storage and system oriented and do not provide a high level of transparency for end-users.

For Network-Enabled Servers (NES) environments [5, 6, 7] (also named Application Service Provider or ASP environments), the need for transparency is crucial since end-users are often

Received
Revised

non computer scientists. In these kinds of environments, servers are deployed over the grid and can be accessed through the classical Remote Procedure Call (RPC) model. Note that GridRPC* provides a standard API for this approach. A NES platform is usually composed of three entities: (1) clients which submit computations to remote servers, (2) servers which are able to process these computations and (3) the agent (the provider) which allows the localization of the most suitable server to solve the problem. In this RPC-based model, data are not supposed to remain available on a server after computation (even if they will be used for a further computation or for another step of the algorithm). Nevertheless, if a client submits a sequence of computations using the same data, it is better to transfer these data only once. More generally, as data already present on a server can be reused for further computations it is necessary to allow the storage of these data locally on the server or at least within the NES architecture. This feature is called *data persistency* and must be as *transparent* as possible for the client.

Data replication is another important feature that a data management service must provide to a NES environment. Indeed, some computations exhibit an inherent parallelism due to independent tasks which process the same data. In this case it is very interesting to replicate these data from a server to another. This allows to overlap communications and computations. This paper presents the data management service called Data Tree Manager (DTM) that we have designed and implemented in the DIET platform [7]. DTM provides a complete data persistency service built on the concepts of data identification and data localization. It also implements a replication service based on performance evaluation to address the choice of the best replica to create and to transfer. DTM is *fully implemented* and experimental results point out the feasibility and the efficiency of this approach. From an end-user point of view, DTM consists in a simple and transparent API fully integrated to the NES API. This paper is organized as follows. The next section gives an overview of related works in data management. Section 3 presents the motivations of our work. It also points out the principles that lead to the definition and the implementation of a data management service. In section 4, we expose the DTM architecture and we justify the different technical solutions that we have adopted. Section 5 describes and comments a set of experimental results which shows the DTM performances and scalability. Finally, section 6 gives a conclusion and a discussion on works in progress in this domain.

2. Related work

In non-NES platforms, several data management approaches exist. One consists in extracting data from their production environment in order to register them in a preservation/management service, while ensuring authenticity, integrity and infrastructure independence. This concept, called *data grid*, focuses on separating data physical view from the logical view to ensure better access and scalability. Many research projects have produced

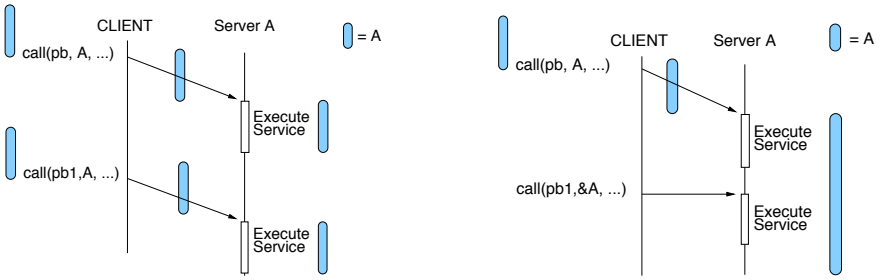
*`https://forge.gridforum.org/projects/gridrpc`

interesting middlewares to cope with this problematic [4, 8]. In the more specific context of Grid computing environments, some data management systems have also been developed. We can cite systems like GASS [9] or LegionFS file system [10]. Here we underline that the NES context is very different from the Grid and Data Grid ones. First, NES environments do not assume a large number of widely distributed nodes. And second, in a NES context, data and applications are loosely coupled while they are tightly coupled in a Data Grid context. In this way and unfortunately, all the previous services are mainly storage and system oriented and are very difficult to adapt to NES environments. Furthermore, in these systems, data transfers are explicitly performed at the client level. This does not fit the transparency needs that we want to provide to a NES client. Another interesting approach is Stork [11]. Stork is a pre-placement tool generally coupled with a meta-scheduler like DAGMan [12] and with a workflow management tool like Pegasus [13]. This approach is quite close to our DTM service but data persistency is managed only during a single computational cycle and again this approach is not transparent enough.

In the NES context, since data mapping greatly influences computations mapping, it is essential to provide a data management service. The aim is to decrease network traffic among clients and servers by ensuring that no unnecessary data are transmitted (and that all necessary data are transferred). At present time, except DIET, only Netsolve [5] provides a data management service. This service is based on two approaches. In the *Request Sequencing* [14] approach when a client submits a sequence of computations, a dependencies graph is produced. Then, the entire sequence is sent to a computational server which first runs the task(s) with satisfied dependencies. When one task (at least) ends, the server updates its dependencies and repeats the same process while there are tasks left to execute. The main drawback of this approach is that data management is performed only for one computation sequence. The *Distributed Storage Infrastructure* [15] approach helps the user to control the placement of data that will be accessed by a server. DSI allows the transfer of data from the client to a storage server. Considering these storage servers closer to computational servers than to the client, the cost of transferring data will be lower when data are reused. In this approach, data transfers are explicit at the client level and useless data transfers between DSI servers and computational servers remain.

3. Motivations and issues

As shown in [16], a NES platform proposes a programming paradigm which is different from other environments such as parallel or distributed environments. Indeed, in NES architectures no data management is supposed to be performed. A data item is not supposed to be available on a server after computations even if it is used for another step of the algorithm. This is the main drawback of NES environments and for this reason it is particularly relevant to propose a data management service.



(a) Sending A twice (b) Sending A only once

Figure 1. Two successive calls

3.1. Use cases

Consider here a simple example where the use of data persistence improves the execution of a GridRPC session. Let assume a client that submits two successive calls with the same input data (see figure 1). With no data management service, the client needs to transfer its input data twice (figure 1.a). Our goal is to provide a service that allows to keep data onto the computational server in order to have a single data transfer (figure 1.b). More generally, our goal is to allow the use of the data already stored inside the platform for later computations and for later sessions. As mentioned in introduction this feature is called *data persistency*.

Molecular Physics Simulation This application simulates the growth of atomic species on a surface [17]. A Kinetic Monte Carlo (KMC) Model is used to simulate the behavior of the atoms depending on the properties of the surface. Input data have a rather small size but the computing phase may be very long (from one day to one month).

The algorithm is composed of two main phases. First, a KMC computation generates a 2D image file. Note that this file can be very large. In a second phase, a ray-tracing process is ran with the previous 2D file image as input data. Now, assuming a NES platform with two computational servers, one providing KMC service and the other providing ray-tarcing service. In this realistic model, the client may be located on the Internet with a high network latency and the servers may be inside the same administrative domain with a low network latency. In the GridRPC standard model, once the 2D image file is generated, it is sent back to the client which has to transfer it again to the ray-tracing server. Here, the interest of a data management service is clearly to transfer the 2D image file from the first computational server to the second one. This will avoid useless data transfers on a high latency network (between the client and the platform).

Dividing Cube The Dividing Cube application [18] consists in extracting and displaying an iso-surface from a three-dimensional medical image. With no data management, the application

can be decomposed in three client calls: (1) the first call aims at extracting an iso-surface and contains a file that may be large (up to 500 MBytes) . It generates a points set in a "surf" format file which is sent back to the client; (2) the second call is the beginning of the visualisation phase and produces a raster format file which is sent back again to the client; (3) finally, this raster format file is sent to the platform in order to be converted into a jpeg file. The main drawback of this application is the number of useless data transfers between the client and the platform. With the use of a data management service, input data could be stored inside the platform and temporary results can be directly exchanged between computational servers. In this example, the number of data transfers is divided by 3.

Replication In [19] a bioinformatic application is presented which runs a large number of independent tasks of short duration. These tasks share large data (from one MBytes to several GBytes) which are regularly updated. As shown in [19] a replication policy may significantly improve the overall performances of the application. In this way, they propose a scheduling and replication algorithm based on the services provided by a NES data management service.

3.2. Issues

Considering the previous use cases, we identify in this section, the issues that a NES data management service must address in order to achieve the goals of persistency and replication.

1. **Data localization** After a data item has been sent once from a client to the platform, the data management service must be able to find where this data item is stored to use it for further computations on other servers.
2. **Data identification** In order to be re-usable, data must be fully identified. A data identifier (or data handle) is a unique reference to a data item that may be stored anywhere inside the platform. By managing data handles, a client does not have to know where its data are currently stored.
3. **Data redistribution** It is a realistic assumption to consider that network bandwidth is better between computational servers than between clients and servers. When data are already stored inside the platform, a data management service must be able to move data between computational servers. Note that when data are moved from one server to an other, their localization information must be updated.
4. **Security** Once data are stored inside the platform, a data manager has to provide a secure access policy. Data can be shared between clients but access rights must be provided. For example, in collaborative projects, a client may want to share its stored data (with other researchers) but does not want anyone to delete them.
5. **Replicas consistency** The consistency issue occurs when some replication operations are performed. In this case, a data item is replicated on several resources. Now, if an update operation occurs on this data item, do all the replicas have to be updated (inducing a large number of transferred messages) ? Or are the replicas viewed as independent copies ? A data management service must provide a reliable and efficient replicas consistency policy.

6. **Data locality** Since manipulated data are often large, it is crucial to minimise the data access cost by applications. In NES environments two kinds of data are used: memory items and files. For memory items, data copy must be avoided. For files items, the copies must be limited to the disk to cache memory copy.
7. **Data Storage** In NES environments, a challenge is to store data as close as possible to computational servers where they will be requested. Since data may be stored during a long time a data management service must cope with physical limitations of storage resources. Otherwise system overload may occur.

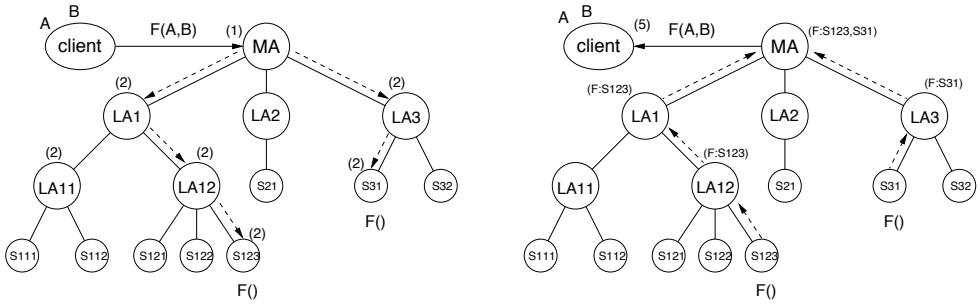
4. The Data Management Service in DIET

In this section, we describe the data management service that we have designed and implemented. This service is currently distributed as a part of the DIET (Distributed Interactive Engineering Toolbox) [7] platform. Nevertheless, since our approach is based on the principles presented in the previous section, the service is flexible enough to be implemented in another NES environment.

4.1. The DIET Platform

DIET is a NES Corba-based multi-agent platform taking advantages of the distributed objects paradigm. One of the main drawback of existing NES platforms is the centralized management of computation invocations by a unique agent that may lead to important bottlenecks. For this reason, DIET proposes a scalable and hierarchical architecture to manage client requests. To identify the role of the different agents, let's assume several servers able to solve the same problem (see figure 2(a) and 2(b)): (1) a client submits a request `F(A,B)` to a `Master Agent` (MA); (2) this MA propagates the client request through its subtrees of `Leader Agents` (LAs) down to the able `Server Daemons` (or SeD: the computational resources). Indeed, since each agent knows which of its child manages the submitted problem, it forwards the client request to them only; (3) each able `SeD` launches the FAST tool [7] in order to estimate the computation time necessary to process the request. This estimation is then sent back to the parent LA; (4) each LA that receives one or more positive answers, selects a pool of servers among the fastest ones. Then, it forwards its answers to the MA, through the hierarchy; (5) once the MA has collected all the answers from its direct childs, it is able to choose a set of fast servers and to send their references to the client. This last one can connect to one of the servers and can send its local data.

In the Grid Computing context, DIET provides access to services (software) and to the related resources (hardware) on a network. Clients submit jobs to servers by asking for the execution of a service: they send their parameters but not the code of the service which is already deployed on the server. For instance, a client asks for a matrix multiplication by sending the type of service (`pdgemm` for example) and the matrices. The programming granularity of this model is coarse grained and job examples may be linear algebraic operations on large matrices as well as other applications or libraries. One of the advantages of this approach is that users do not need to be experts in parallel programming to get the benefit of high



(a) Forwarding a request

(b) Aggregating the responses

Figure 2. Forwarding/Aggregating requests and responses in a hierarchical search in trees of LAs

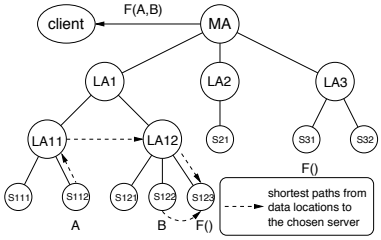


Figure 3. Problem submission

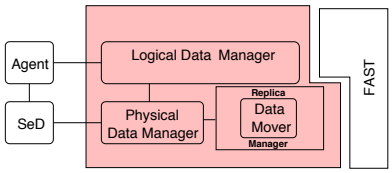


Figure 4. The DTM architecture

performances parallel programs and computers. This approach differs from the Globus [1] or Legion [10] approach which hides the underlying hardware to the programmers and users by providing uniform access to the resources. In this case, programmers are usually experts in parallel programming.

Now, assume that for the same problem, each operand needed for the computation is already available in the platform and on distinct servers (see figure 3). In this example and as previously exposed, a data management service must be able: (1) to re-use the already stored data (`A` and `B`); (2) to choose the best server by estimating data transfer time to this server; (3) to transfer the right data at the right place. These features must be provided to end-users via the most simple and transparent API.

4.2. The Data Tree Manager architecture

The data management service we have designed and implemented is called DTM (Data Tree Manager). Our goals were to minimize the cost of adding such service while ensuring good

scalability and efficiency. In order to avoid interlacing between data messages and computation messages, the policy we propose separates data management from computation management (see figure 4). In this way, our service is composed of four components:

1. **The Logical Data Manager** The Logical Data Manager is composed of a set of `LocManager` objects. A `LocManager` is linked to the agent (MA or LA) with which it communicates locally. It manages a list of tuples `(data identifier, owners)` which represents all the data present in its sub-tree. Clearly, the global hierarchy of `LocManager` objects provides the *localisation* knowledge of the data managed by the platform.
2. **The Physical Data Manager** The Physical Data Manager is composed of a set of `DataManager` objects. A `DataManager` is located on each `SeD` with which it communicates locally. It owns a list of `persistent` data. It stores data and has to provide them to the server when needed. It informs its parent `LocManager` of updates operations performed on its data (add, move, delete) and it also provides functionalities for inter-servers data movements.
3. **The Data Mover Protocol** The Data Mover protocol provides mechanisms for data transfers between `Data Manager` objects. These transfers are not performed by `SeD` objects to avoid to overload them by data management. Furthermore, in order to integrate different transfer protocols, such as GridFTP or RFT, we have chosen to separate data transfer management and data recording management (by `DataManager` objects). An orthogonal role of the `Data Mover` is also to initiate updates of `DataManager` and `LocManager` as soon as a data transfer is completed.
4. **Replica Manager** The `Replica Manager` aims at sending replication orders to the `Data Mover`. It allows the choice of the best replica to be transferred when replication operations occur. This choice is based on network forecasts information provided by the NWS tool. Files or memory items are replicated and a distributed protocol is used (there is no more distinction between the original data item and its replicas). At the present time, we use a `Replica Manager` instance in which replicas are read-only but it is possible to write other instances implementing other consistency policies. Note that the replica consistency techniques used in distributed databases [20] cannot be directly applied here since the execution context is different from the NES context (heterogeneity, knowledge of data structure, physical distance between replicas, etc).

Several advantages arise from a such hierarchy. First, at the decision level (the DIET Master Agent), all the platform information are collected. So, at the same level, it is interesting to provide a localization service that will be able to store all needed information about data managed by the platform. Second, communications are always local exchanges between DIET and DTM components. This induces a low bandwith consumption by data management processes. Third, the hierarchical architecture allows updates operations to be limited to only a subtree of the platform. Again, communication overheads are minimised. Finally and due to the use of CORBA, DTM minimises the number of data copy operations. This optimisation is a crucial point when data are large. Thanks to buffers management mechanisms, CORBA allows to share memory space between objects. Note that the use of CORBA, also ensures physical and logical infrastructure independence.

4.3. The end-user point of view

Only end-users have the knowledge of the applications they submit to the platform. Therefore, only end-users have the knowledge of the data that must be managed or not by the platform. Taking this into consideration, we propose a new set of functions and a persistence flag to support data management. In this way, a client can choose whether a data item must be persistent inside the platform or not. This property is called the `persistence mode` of a data item. Based on our experience gained on use cases we define five data persistence modes: DIET_VOLATILE (data are not persistent), DIET_PERSISTENT (data are stored on servers and movable between servers), DIET_PERSISTENT_RETURN (data are stored on servers, movable and a copy is sent back to the client), DIET_STICKY (data are stored but not movable between computational servers) and DIET_STICKY_RETURN (the same as DIET_STICKY plus data are sent back to the client). Furthermore, when a data item is stored inside the platform an identifier is assigned to it. So a client must have the knowledge of this identifier in order to (re-)use the corresponding data. A client only knows the identifier of persistent data it has generated and he is responsible for propagating this information to other clients. By using data handles, clients do not need to know where data are stored.

The proposed API is based on the *profile* concept. A profile is built by a client. It is composed of the submitted problem name and of all the needed data and their persistence mode. For data that are already managed by the platform, data are replaced by their handles. A client can perform the following operations (for more details the reader can refer to [21]): (1) **Recording identifiers** in a local client file. This will be helpful to use these data in other sessions of the same client or of other clients; (2) **Using data** already stored in the platform and identified by a handle; (3) **Removing data** to free persistent data identified by a handle; (4) **Reading an already stored data** identified by a handle

4.4. DTM functioning

As presented above the DTM structure is mapped onto the DIET one. Therefore, the DTM initialisation follows the hierarchical building of DIET (in a descending order).

Adding a data When a data item is added to a `DataManager`, the localization information update is performed in the ascending order of the hierarchy. These update phases follow a client request which transfers its data to a `SeD` for computation. Adding a new persistent data item requires three different steps as presented in figure 5. First, the data item is registered on the `DataManager`. Then, its identifier and properties are transfered to the parent `LocManager` object. Finally, the `LocManager` registers these information with the source of the data item (i.e. the `DataManager`) and informs its parent `LocManager` object. This last step is repeated until the root `LocManager` is reached.

Deleting data When a data item is deleted from a `DataManager` (following a delete call by the client) the localization update is performed in the descending order of the hierarchy. This ensures that the data item reference is deleted from the platform as soon as the remove call is sent out (even if the data item is not physically deleted). This way, in a first step, the

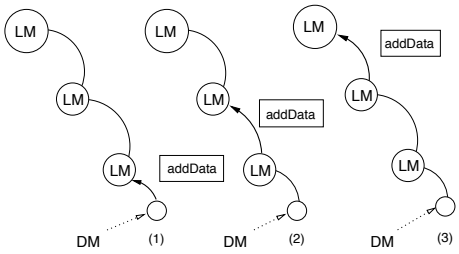


Figure 5. Adding data

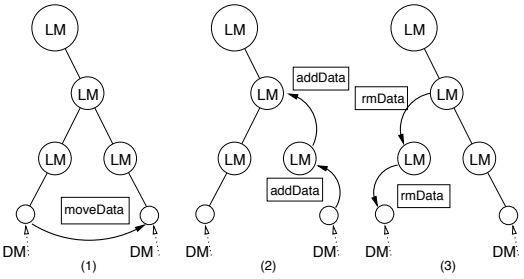


Figure 6. Moving data

`LocManager` receives the deleting order from the `Master Agent`. Then it deletes the associated reference and contacts its child (belonging to the sub-tree holding the data item) which iterates the operation until a `DataManager` is reached. This last one is then able to delete the data item and its properties. Note that an occupation flag is used for persistent data. When this flag is on for a data item, no deleting operation can be performed on it.

Looking for data Let's assume a client request with a data item already present in the platform. As exposed above the client only provides the identifier of the data. When a `SeD` receives the request, it initiates the search of the data item within the platform. The search algorithm is exposed in [21] and is performed by the `DataManager`. When the data item is found, a moving procedure is ran (see below).

Moving and replicating data When a data item is moved from a `DataManager` to another `DataManager`, only a part of the hierarchy is updated. In a first step the data item is moved thanks to the `DataMover`. As shown figure 6, the next steps consist in an "adding" and a "deleting" step. Note that for a replication operation, the "deleting" step is not required. The update operation is critical since an utilisation request may occur during the data transfer. In order to solve this problem, the update is initiated by the `DataManager` which receives the data item. In this way, a data item is always localised even if the localisation is no more valid. If a `DataManager` receives a moving request for a data item that it does not own, it initiates itself a "search" step (as previously presented).

Security and fault tolerance issues An access key is added to the data identifier. In this way, if a client wants read/write rights on a specified data, it has to join this key to the data identifier. Furthermore, DTM does not define fault recovery mechanisms but only a consistency mechanism of the infrastructure when faults occur. We ensure that all operations (add, delete, move) made on data by clients are made to guarantee the consistency of the infrastructure. For more details, reader can refer to [21].

Data size (Mo)	0.1	0.2	0.5	0.75	1	1.5	2	5	10	20	30
Without DTM (s)	27.76	28.96	32.97	36.4	39.7	46.26	52.93	92.71	159.31	291.57	424.31
With DTM (s)	28.13	29.69	33.86	37.49	41.54	47.82	54.85	94.81	162.51	294.41	428.92
Overhead (in %)	1.33	2.5	2.69	2.99	4.63	3.37	3.62	2.27	2.0	0.97	1.08

Table I. DTM overhead

5. Experimental results

We have conducted a set of experiments in order to demonstrate the performances of the DTM service. These tests show: (1) the low cost of the DTM infrastructure; (2) the scalability of the platform; (3) the relevance of the data persistency approach; (4) the performances of the data replication policy. Finally we illustrate these benefits running the Dividing Cubes use case. In the first set of experiments, the DIET platform is deployed over a local architecture on the VTHD high speed network (1 GigaBit/s). The second platform is deployed over two laboratories distant from about 100 kilometers.

5.1. DTM overhead and scalability

The aim of the first experiment is to evaluate the overhead due to the use of DTM. 26 clients submit synchronously 10 times the same problem to 26 different servers with various file sizes: (from 10 KBytes to 30 MBytes). The experiment is performed with and without DTM. Table I shows the SeD computation time along with data size variation. It clearly shows that the overhead due to the use of DTM is low (an average of 2.36%).

The aim of the second experiment is to show the behaviour of our system in the case of an important load of requests. This experiment has been performed on a DIET architecture composed of 1 MA, 2 LAs and 25 SeDs. 26 clients submit a number of requests which varies from 1 to 550 (for all clients). Each data size is 100 MBytes. It appears that the global servers latency (the time the server waits between two requests) is lower with data management than without: of about 55% for 500 requests (see figure 7).

5.2. Data persistence benefits

Here, the target architecture is composed of 1 MA, 2 LAs and 2 SeDs. These elements are interconnected through a local network and a client is located in a remote site 100 kilometers far away from the DIET platform. The interconnection network is a 16 Mbits/s network while the local area network is an Ethernet 100 Mbits/s network. The deployed application is a linear algebra application with computation times rather independent from data size (meantime 50 seconds).

The aim of this experiment is to show the benefit of using DTM in the case where input data are used several times by several computations. As shown figure 8(a), the execution time varies a lot according to the cases. When data are persistent and locally stored on the computational server, the global execution time is equal to the application computation time. The difference between the “with DTM” and the “without DTM” case is important. It is approximately of

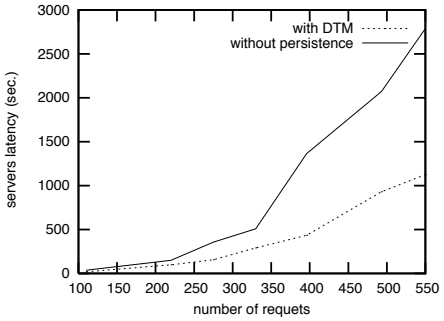
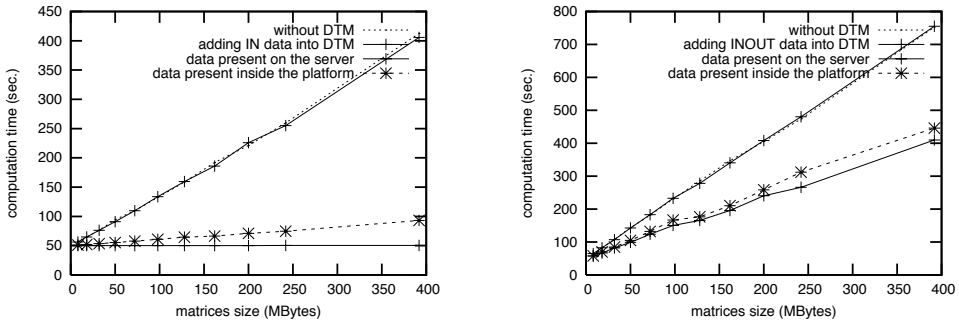


Figure 7. Servers latency



(a) Sending input Data

(b) Sending input/output data

Figure 8. Sending input or input/output data

87% for a 400 MBytes matrix and corresponds to the data transfer time from the client to the servers. When data are moved between computational servers (the "data present inside the platform" curve in the figure) the profit is of an order of 77% for a 400 MBytes matrix. Again, the difference corresponds to the data transfer time. Now, in the case of input/output data, gains are less important than for the first experiment (see figure 8(b)). An order of 45% is reached for a 400 MBytes matrix if data are local to the computational server and 40% if data are moved. Even if these results are not generally applicable (the gains are closely linked to the experimentation conditions), they confirm the feasibility and the efficiency of our approach. Note that interested readers can find comparison results between NetSolve and DTM in [21].

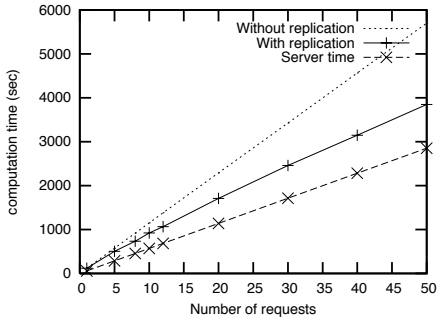


Figure 9. Replication results

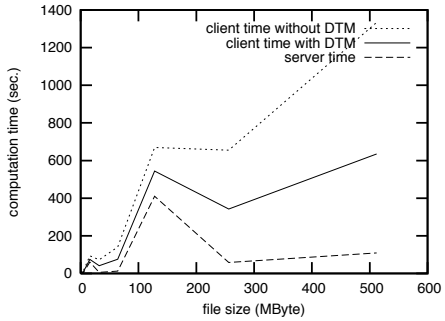


Figure 10. Dividing Cubes results

5.3. Replication benefits

The architecture deployed for these experiments is composed of one MA and 6 SeD. The interconnection characteristics are the same as in the previous experiments. The application consists in computing the occurrences number of a letter in a file (of 50 Mbytes size). A set of synchronous requests is sent to the platform and when a data item required by a server is not present then it is replicated. Results are presented figure 9. The "with replication" graph shows a linear progression in the beginning (until the sixth request) and then inflects its progression. This can be explained by the fact that until the sixth request, the input file is replicated on the different servers. Then, for next requests, each server owns the data file and no more data transfers are required.

5.4. Illustration with the Dividing Cubes algorithm

In this section, we illustrate the benefits of the DTM service on the implementation of the Dividing-Cubes algorithm described in section 3.1. The target architecture is built as follows: the client and the agent are located at the same place and the server is 100 km far away. The WAN is a 16 shared MBytes/s network. The experiment consists in input files of various sizes (from 0.1 Mbytes up to 500MBytes) on which several extraction parameters are applied. The result is a jpeg file that is sent back to the client.

The figure 10 exhibits the advantages of using DTM. When the data management is not used, a client has to send its input file each time it wants to run an extraction. This is particularly penalising when only some parameters need to be updated for the extraction. Note that, as input files are automatically generated with various characteristics, the computational time increases abnormally for input files sizes of respectively 16Mbytes and 128 MBytes. However, the main consideration remains that the global computation time is consistent. These results exhibit that the larger the input files are, the more useful DTM is: a benefit of 52% for a 512 Mbytes file size.

6. Conclusion and future works

In this paper we describe DTM, an efficient data management service for NES environments. This service is *fully implemented and distributed* with the DIET platform. Based on the concepts of data persistence and data replication, DTM allows to minimise computation times by decreasing data transfers between the clients and the platform. As end-users of NES platforms are often non computer scientists, DTM provides a simple and transparent API. A series of experiments has been conducted and demonstrates the feasibility and the efficiency of our approach (in terms of performance gains and scalability). Note that, in the context of normalisation, we are currently making a proposal in the GridRPC Working Group in the Global Grid Forum[†]. This proposal aims at standardizing a data management API for GridRPC platforms.

In our future works, we plan to extend our tests to the Grid'5000 experimental platform. This platform provides an interesting opportunity to test DTM at limit conditions and with a large number of nodes distributed over France. We expect to show the efficiency and the robustness of our choices on a large scale context. Our future tests campaign also include the integration of DTM in the GriPPS[‡] application (see also section 3.1). This integration is quite finished and tests will follow. Now, we are currently working on providing to clients the ability to use external data. For instance, these data can be data stored on data grids depots such as IBP or SRB. A client will then be able to specify data sources and results destinations. For this, we are extending `Data Mover` to other protocols such as GridFTP. We see several advantages to this approach. First, it will possible to link the DTM data life cycle to the one used by data grids. Second, a client will not have to recover locally a data item before to send it to the DIET platform. A single data transfer will thus be performed.

Finally, we end this paper by replacing our work in the Data Grid context. In this context, the data management is mainly storage and system oriented and is tightly coupled to the application part. In our case, we clearly separate the data management from the application management. However, it seems to be possible to link DTM with Data Grids systems. Ninf with Ninf-G and Netsolve with Netsolve-G propose a such service at the application level. A similar solution at the data management level would allow to transfer data between grid applications with DTM. In this way, data transfers could be performed in a transparent way since only "in data" would have to be explicitly known. Then, a grid application developer will no longer have to focus on data localization since DTM will bring data to applications when needed.

REFERENCES

1. Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[†]`https://forge.gridforum.org/projects/gridrpc-wg/`
[‡]`http://gripps.ibcp.fr/`

2. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation - Practice and Experience*, 17(2-4):323–356, 2005.
3. W. Hoschek, F.J. Janez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *GRID*, pages 77–90, 2000.
4. A. Chervenak et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets, 1999. `http://www.globus.org`.
5. K. Seymour, A. Yarkhan, S. Agrawal, and J. Dongarra. Netsolve: Grid Enabling Scientific Computing Environments. In L. Grandinetti editor, editor, *Grid Computing and New Frontiers of High Performance Processing*, volume 14. Elsevier, Advances in Parallel Computing, 2005.
6. Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1):41–51, 2003.
7. E. Caron et al. A Scalable Approach to Network Enabled Servers. In *EuroPar 2002*, volume 2400 of *LNCS*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.
8. G. Singh and et al. A Metadata Catalog Service for Data Intensive Applications. In *Proc. of SC2003 Conference*, November 2003.
9. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *6th Work. on Input/Output in Par. and Dist. Sys.*, pages 78–88, Atlanta, GA, 1999. ACM Press.
10. B.S. White, M. Walker, M. Humphrey, and A.S. Grimshaw. LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications. In *Proc. of the IEEE/ACM Supercomputing Conference (SC2001)*, November 2001.
11. T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid, 2004.
12. Condor Team University. Condor version 6.1.12 manual, 2005.
13. J. Blythe et al. Task Scheduling Strategies for Workflow-based Applications in Grids. In *CCGrid 2005*, Cardiff, 2005.
14. D.C. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. *LNCS*, 1900:1213, 2001.
15. M. Beck and et al. Middleware for the Use of Storage in Communication. *Parallel Computing*, 28(12):1773–1788, December 2002.
16. F. Desprez and E. Jeannot. Improving the GridRPC Model with Data Persistence and Redistribution. In *3rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Cork, Ireland, July 2004.
17. F. Picaud, C. Ramseyer, C. Girardet, and P. Jensen. Confinements effects on the growth of adsorbates: Interpretation of the formation of monoatomic Ag wires on Pt(997). *Physical Review*, 61(23):154–162, 2000.
18. S. Miguet and J.-M. Nicod. Complexity analysis of a parallel Marching-Cubes. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(7):1024–1041, 1997.
19. F. Desprez and A. Vernois. Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid. In *CLADE 2005*, Research Triangle Park, NC, July 2005. IEEE Computer Society Press.
20. S. Gancarski, C. Le Pape, and H. Naacke. Fine-grained Refresh Strategies for Managing Replication in Database Clusters. In *VLDB Worshop on Design, Implementation and Deployment of Database Replication*, Norway, 2005.
21. B. Del-Fabbro. *Contribution à la gestion des données dans les grilles de calcul à la demande : de la conception à la normalisation*. PhD thesis, UFR des Sciences et Techniques de l'Université de Besançon, December 2005.